論文2003-40SD-8-10

효율적인 혼합 BIST 방법

(A Newly Developed Mixed-Mode BIST)

金 玄 炖 *, 愼 鏞 升 *, 金 容 準 *, 姜 成 昊 *

(Hyundon Kim, Yongseung Shin, Yongjoon Kim, and Sungho Kang)

요 약

테스터를 사용하는 테스트 방법이 매우 비싸고 동작속도에서의 테스트가 어려운 상황에서 BIST의 출현은 이러한 난점을 해결하는 좋은 방법이다. 하지만, 이러한 BIST에도 해결해야 할 문제점들이 많다. 의사 무작위 테스트시 패턴 카운터와 비트 카운터의 역할이 단순히 카운팅만 하는데 한정되어 있으므로 이들 카운터를 패턴을 생성하는 역할에도 이용함으로써 BIST의 효율을 증대시키고자 한다. 새로운 BIST 구조는 LFSR이 아닌 카운터로 패턴을 생성하고 LFSR로 이의 동작을 무작위하게 또는 의도적으로 조정함으로써 다른 테스트 성능의 저하 없이 테스트 하드웨어를 축소하는 방법을 제안한다. 결정 테스트를 위한 하드웨어가 너무 크게 되는 단점을 해결하고자 본 논문에서의 실험은 실험결과에서 의사 무작위 테스트와 결정 테스트의 성능을 고장검출율, 테스트 시간과 하드웨어 관련 인자들로 표현한다.

Abstract

Recently, many deterministic built-in self-test schemes to reduce test time have been researched. These schemes can achieve a good quality test by shortening the whole test process, but require complex algorithms or much hardware. In this paper, a new deterministic BIST scheme is provided that reduces the additional hardware requirements, as well as keeping test time to a minimum. The proposed BIST (Built-In Self-Test) methodology brings about the reduction of the hardware requirements for pseudo-random tests as well. Theoretical study demonstrates the possibility of reducing the hardware requirements for both pseudo-random and deterministic tests, with some explanations and examples. Experimental results show that in the proposed test scheme the hardware requirements for the pseudo-random test and deterministic test are less than in previous research.

Keyword : Built in Self Test, deterministic test pattern, LFSR, reseeding

I. 서 론

내장형 자체 테스트 기법(BIST : Built in Self Test)은 점점 더 거대해지고 있는 회로를 적은 비용과 동작속도로 테스트하기 위한 효과적인 방법으로 주목받고 있는 방법이다[1].

BIST의 효율성은 테스트 시간과 테스트 하드웨어, 고장검출율로 나타낼 수 있다. 많은 BIST 구조가 의사 무작위 테스트, 재초기화 방법, 가중치 의사 무작위 테스트, 결정 테스트 방법 등에 관해 이러한 parameter들의 절충점을 찾기 위해 연구되어 왔다[2-7, 11-18].

본 논문은 의사 무작위 테스트와 결정 테스트를 위해 효율적인 구조와 알고리즘을 제시한다. 이 두가지

* 正會員, 延世大學校 電氣電子工學科
(Dept. of Electrical Eng., Yonsei Univ.)

※ 이 연구는 2000년도 한국과학재단 연구비 지원에 의한 결과임.(과제번호 : 2000-1-30200-002-3).
接受日字:2002年10月30日, 수정완료일:2003年7月31日

테스트를 모두 하는 것을 혼합모드 테스트라고 한다. 의사 무작위 테스트는 일반적으로 LFSR(Linear Feedback Shift Register)을 이용하여 의사 무작위 패턴을 생성하는 것인데, 테스트 벡터를 생성하는데 큰 하드웨어를 필요로 하지는 않지만, 높은 고장검출율을 보장할 수 없는 단점이 있다. 대부분의 큰 회로는 의사 무작위 패턴만으로는 100% 고장검출율을 얻지 못하는데, 이 때 검출되지 않는 고장을 검출 난해 고장이라 한다. 검출 난해 고장의 수를 줄이거나 그들의 검출을 용이하게 하기 위해 재초기화 방법[16-18]과 가중치 의사 무작위 패턴 생성 방법[13-15]과 같은 의사 무작위 테스트의 변종 테스트 방법이 사용되고 있다. 결정 테스트는 디코딩 로직[3, 4, 10]이나 ROM[5, 7]을 이용하여 작은 크기의 결정 패턴 집합을 생성한다. 그러나, 결정 테스트는 매우 큰 하드웨어를 필요로 하기 때문에 의사 무작위 테스트로 고장검출율을 적당히 올린 후 검출 난해 고장에 대해서만 결정 테스트를 하여 고장검출율을 올리게 된다.

기존의 의사 무작위 패턴 생성 방법은 단일 스캔체인을 사용한다고 가정할 경우, LFSR로 패턴을 생성해서 그 중 하나의 값을 계속해서 스캔체인으로 보내는 형태이다. 따라서, LFSR의 단 수가 n이라 할 때, 스캔체인의 길이에 상관없이 $2^n - 1$개 만큼의 패턴을 생성할 수 있다. 이를 위한 다른 하드웨어로는 비트 카운터(혹은 쉬프트 카운터)와 패턴 카운터를 들 수 있다. 비트 카운터의 길이는 스캔체인의 길이에 의해 결정되는데, 스캔 체인의 길이가 m이라 하면, $\lceil \log_2 m \rceil$ 만큼의 길이를 갖는 비트 카운터를 사용한다. 비트 카운터는 스캔 당 테스트를 하는 경우, 스캔체인에 쉬프트 과정이 끝나고 테스트 대상 회로로부터 응답을 받는 시점을 알려주는 역할을 한다. 테스트가 끝나는 시점을 알려주고, 결정 패턴 생성시 컨트롤을 위해 필요한 것이 패턴 카운터인데, 현재 몇 번째 패턴이 회로에 가해지고 있는지를 나타내는 색인의 역할을 한다고 할 수 있다. 즉, 스캔체인에 한 번 값이 다 채워져서 테스트 대상 회로에 패턴이 가해질 때마다 값이 하나씩 커지는 형태의 카운터로서, 결정 패턴 생성시 언제 비트 플립핑(bit-fliping)을 할지 결정하는데 기준이 되는 역할을 하는 부분이다.

본 논문에서는 패턴 생성기의 구조를 조금 변형하여 더 작은 하드웨어로도 충분한 의사 무작위 패턴을 생성하도록 하였다. 본 논문은 LFSR 대신 카운터에서 스캔 체인으로 값을 밀어넣고 스캔 체인으로 들어갈 값의 선택을 작은 LFSR이 하게 된다.

재초기화 방법[16-18]은 반복적인 시뮬레이션을 통해 효과적인 초기값을 찾아낸다. 많은 경우, 결정 패턴은 "auto-correlated"되어 있다("auto-correlated"라 함은 하나의 결정 패턴이 또 다른 하나의 결정 패턴이 쉬프트 된 형태의 패턴이라는 것이다.). 따라서, 이러한 경우, 어떤 결정 패턴이 생성되면, 많은 다른 결정 패턴이 쉽게 생성될 수 있다.

가중치 의사 무작위 테스트[13-15]는 0 혹은 1의 수가 고르게 있지 않고 쏠려 있는 패턴을 테스트하는데 효과적인 테스트 방법인데, 한 쪽으로 쏠려 있는 정도가 심할수록 좋은 결과를 얻을 수 있는 방법이다.

비트 픽싱[10]은 매우 직관적인 방법으로서, 어느 시점에 어느 패턴을 넣어야 하는지를 LFSR이나 카운터들의 값을 참조하여 생성하는 방법인데, 알고리즘이 간단한 반면 요구되는 결정 패턴의 수가 많은 경우 하드웨어의 부담이 상당한 방법이다.

다중 다항식을 갖는 LFSR을 이용한 혼합모드 스캔당 테스트 구조가[19]에 있는데, 결정 패턴의 효율적인 인코딩을 목적으로 하였으며, 많은 결정 패턴이 무상관(don't care) 값을 포함하고 있다는데 착안한 논문이다.

혼합모드 패턴 생성의 대표적 구조가 <그림 1>과 같다. <그림 1>에서 디코딩 로직은 PRPG(Pseudo-Random Pattern Generator)와 카운터의 값을 참조하여 스캔 체인으로 들어갈 값을 직접 제어한다. 즉, PRPG에서 생성되는 의사 무작위 패턴이 결정 패턴과 일치하는 경우 0을 출력하고, 결정 패턴과 다른 경우 1을 출력하여 결정 패턴을 생성하는 방법이다. 따라서, 디코딩 로직의 크기는 거의 절대적으로 디코딩 로직이 1을 출력해야 하는 횟수에 의존한다. 즉, 의사 무작위 패턴

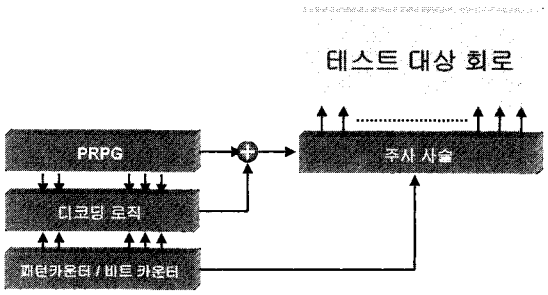


그림 1. 패턴 카운터와 비트 카운터의 쓰임
Fig. 1. Usage of a pattern counter and a bit counter.

이 결정 패턴과 비슷하면 비슷할수록 더 작은 크기의 디코딩 로직으로도 결정 패턴을 생성해낼 수 있다. 본 논문에서의 결정 패턴 생성 방법도 이러한 부분에 초점을 맞추어 연구되었다.

제안하는 방법은 의사 무작위 테스트와 결정 테스트를 위한 하드웨어를 줄이면서도 고장검출율이나 테스트 시간의 측면에서의 성능 저하가 전혀 발생하지 않는다.

본 논문에서는 새로운 혼합모드 테스트 방법이 제 2장에서 소개되었고, 이는 의사 무작위 패턴 생성과 결정 패턴 생성 부분으로 나뉘어 소개되었다. 실험결과는 제 3장에서 소개되었고, 제 4장에서 새로운 방법의 성능을 정리하였다.

Ⅱ. 새로운 혼합모드 테스트 방법

1. 의사 무작위 패턴 생성

제안하는 구조는 LFSR의 한 비트가 스캔체인으로 들어가는 형태와 달리, 원래 사용해야 하는 구조인 패턴 카운터와 비트 카운터의 여러 비트중 한 비트를 골라서 스캔체인으로 보내는 구조여서, 한 자리에서만 스캔체인으로 계속 값을 보내는 것이 아니며, 스캔체인으로 들어갈 값을 선택하는 역할은 작은 LFSR이 맡는다. 여기서는 5비트의 LFSR을 이용하였다.

새로운 구조의 의사 무작위 패턴 생성 방법은 <그림 2>와 같다.

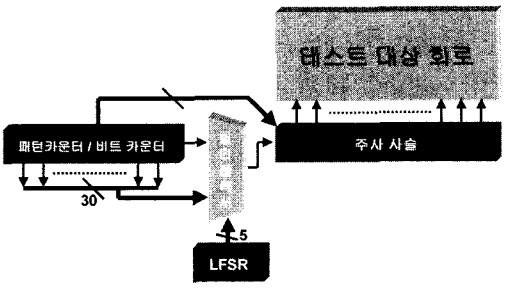


그림 2. 제안하는 의사 무작위 패턴 생성 구조
Fig. 2. The proposed structure for pseudo-random pattern generation.

<그림 2>에서 보이는 구조는 5비트의 LFSR이 32비트의 [패턴 카운터 / 비트 카운터]에서 총 32비트중 한 비트를 제외한 31비트중 한 비트만 스캔체인으로 보낼 수 있도록 패턴을 생성한다. 32비트 모두가 아닌 31비트만이 멀티플렉서로 들어가는 이유는 5비트의 LFSR이 생성할 수 있는 패턴의 가지 수가 모두 0인 패턴(혹은 모두 1인 패턴)을 제외한 2^5-1가지이기 때문이다. <그림 3>에서 패턴 카운터와 비트 카운터는 각각 일반 카운터(값이 1씩 올라가는 카운터) 혹은 LFSR 형태로 구성할 수 있으며, 비트 카운터는 일반 카운터의 형태를 유지하기로 한다. 뒤에 나오는 결과 테이블에서는 비트 카운터를 일반 카운터의 형태뿐 아니라 LFSR 형태인 경우의 결과도 포함하였다. 패턴 카운터의 경우는 비트 카운터와 함께 일반 카운터를 쓰는 경우 LFSR의 형태로 쓰는 경우보다 의사 무작위 테스트에서 고장검출율이 조금 낮아서 LFSR의 형태로 사용했고, XOR 게이트가 각 단의 외부에 있는 타입의 LFSR이 아닌 XOR 게이트가 내부에 있는 타입의 LFSR의 형태를 사용하였다. 5비트의 LFSR에서 패턴 카운터와 비트 카운터의 값을 선택할 때, 0 또는 1의 값중 한 쪽의 값이 너무 많고 적을 경우, LFSR에서 선택하는 값이 0 또는 1쪽으로 몰릴 것이고, 그러한 패턴의 특성이 전체 고장검출율을 빠른 시간 안에 올리는데 제한이 될 것이기 때문에, 패턴 카운터로서 한 클럭 주기에 0 또는 1의 수가 하나씩만 달라지는 타입의 out-tapped LFSR을 쓰기보다는 그 수가 여러개씩 바뀔 수 있는 타입의 in-tapped LFSR을 사용하는 것이다.

이 구조가 기존의 구조보다 적은 수의 비트를 사용하고도 같은 수준 혹은 그 이상의 고장검출율을 유지할 수 있는 것은 의사 무작위 패턴 테스트를 하는 동안의 패턴 카운터와 비트 카운터의 역할을 증대시켰기 때문이다.

<그림 1>와 <그림 2>에서 패턴 카운터와 비트 카운터의 총 비트 수를 같게 놓으면 <그림 2>의 LFSR 길이가 더 짧다. <그림 2>에서의 LFSR 길이를 L이라

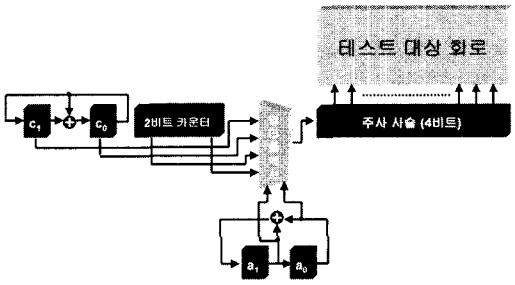


그림 3. 제안하는 방법의 패턴 생성의 간단한 예
Fig. 3. A simple example of the proposed pattern generation.

표 1. 그림 3의 패턴 생성기로부터의 패턴
Table 1. Patterns from the pattern generator in Fig. 3.

a_1	a_0	c_1	c_0	2비트 카운터		주 사 사 슬			
0	1	1	1	0	0	x	x	x	x
1	0	1	0	0	1	1	x	x	x
1	1	0	1	1	0	0	1	x	x
0	1	1	1	1	1	0	0	1	x
1	0	1	0	0	0	1	0	0	1
1	1	0	1	0	1	0	1	0	0
0	1	1	1	1	0	1	0	1	0
1	0	1	0	1	1	1	1	0	1
1	1	0	1	0	0	1	1	1	0
0	1	1	1	0	1	0	1	1	1
1	0	1	0	1	0	1	0	1	1
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.

하면, <그림 1>에 보이는 기존의 구조는 LFSR이 2^L의 길이를 가지므로 $2^L = m$이라 하면, $2^m - 1$의 패턴 생성 길이를 갖고, <그림 2>에 보이는 구조는 $(m-1)\times 2^m$ 정도의 패턴 생성 길이를 갖는다. 따라서, 이론적으로도 <그림 2>의 구조가 <그림 1>보다 의사 무작위 테스트에서 효과적인 테스트를 할 수 있음을 알 수 있다.

이를 위한 간단한 예는 <그림 3>과 같다.

<그림 3>과 <표 1>에서 보이는 패턴 생성기는 간단한 예로서, 외부 XOR 타입의 2비트 LFSR, 패턴 카운터로서 내부 XOR 타입의 2비트 LFSR, 비트 카운터로서 2비트의 카운터를 이용하고, 스캔체인의 길이가 4인 경우이다. 실제의 경우에는 스캔체인의 길이를 h라 했을 때, 비트 카운터가 $\lceil \log_2 h \rceil$ 비트만큼의 길이로서 스캔체인보다는 훨씬 짧은 길이가 되는데, 여기서는 간단한 예를 보이려다 패턴 카운터와 비트 카운터의 총 비트 수와 스캔체인의 길이가 같게 구성되어서, 실제처럼 스캔체인에서 다양한 패턴이 생성되지 않을 수도 있다. <표 1>에서 보여주는 패턴의 생성 주기는 시작 벡터 ($a_1a_0c_1c_0$[2비트카운터]) = 011100로부터 그것이 다시 반복되는 간격인 $(4-1)(2^2-1)\times 2^2 = 36$이 되어야 하는데 여기서는 LFSR의 주기가 3이고 카운터의 주기가 12이기 때문에 서로 소가 아니어서 패턴 생성 주기가 12가 된다. 하지만, 이러한 경우는 LFSR과 카운터를 구성함에 따라 충분히 피해갈 수 있는 경우이므로 큰 문제는 되지 않는다 할 수 있다.

2. 결정 패턴 생성

결정 패턴 생성을 위한 새로운 알고리즘은 <그림 4>와 같다.

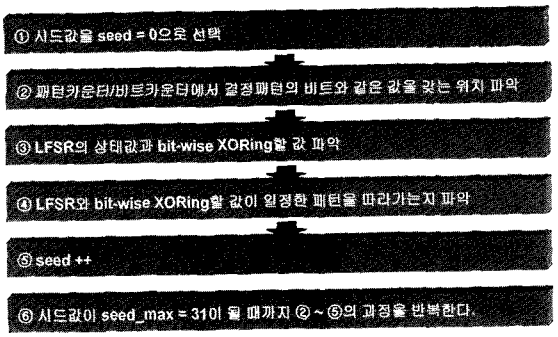


그림 4. 결정 패턴 생성을 위한 알고리즘
Fig. 4. An algorithm for deterministic pattern generation.

① 시드값을 seed = 0으로 선택

여기서, 시드값이라 함은 비트 카운터의 초기치를 말하며, 이 다섯 비트의 값은 00000에서 시작하여, 매 클럭마다 값이 하나씩 올라가며, 11111까지 간 다음 클럭에서의 값은 00000 이런식으로 값이 바뀐다.

② 패턴카운터/비트카운터에서 결정 패턴의 비트와 같은 값을 갖는 위치 파악

패턴카운터와 비트카운터를 포함하여 32비트인데, <그림 2>에서도 알 수 있듯이 이 중 하나의 값을 LFSR에서 선택하는 형태이므로, 32비트 중 그 시점에 해당하는 결정 패턴의 비트 값과 동일한 값을 갖는 스테이지를 파악한다.

아래와 같은 <그림 5>에서 패턴카운터는 24비트 LFSR을 포함하며, 비트카운터는 8비트 카운터만 포함한다. 만일 스캔체인이 더 작은 길이의 비트카운터를 필요로 하거나, 더 긴 비트카운터를 필요로 하면, 그만큼 비트카운터의 길이를 조정하며, 24비트 LFSR이라고 되어있는 부분의 값을 반대로 조정하여, 패턴카운터와 비트카운터의 전체 길이를 32비트로 고정한다.

패턴카운터와 비트카운터가 <그림 5>와 같은 구조를 갖는다는 가정 하에 패턴카운터/비트카운터에서 결정 패턴의 비트와 같은 값을 갖는 위치를 파악하는 방법의 예를 들면 다음 <그림 6>과 같다.

패턴카운터/비트카운터의 가장 오른쪽에 있는 비트부터 0번, 1번, …, 31번이라 하고, 결정 패턴이 연속된 세

개의 1을 필요로 한다면, 첫 번째 상태값에서는 1,4,7,9,10,12,30,31이 결정 패턴과 매치되는 값이다. 마찬가지로, 두 번째, 세 번째 상태값에서는 각각 0,1,4,7,8,9,10,12,29,30, 2,4,7,11,12,28,29,31이 결정 패턴과 매치되는 값이다.

③ LFSR의 상태값과 비트별 XORing할 값 파악

<그림 6>에서의 결정 패턴과 매치되는 값을 선택하기 위해 LFSR의 상태값과 XORing할 값은 아래 <그림 7>과 같이 구한다.

<그림 7>에서 LFSR 왼쪽에 표시된 값은 그 시점에서의 LFSR의 상태값이고, 그 오른쪽에 괄호 안에 있는 값은 ②의 과정에서 파악된 값이고 괄호와 함께 표시된 값은, 그 파악된 값을 선택하기 위한 멀티플렉서의 컨트롤 신호를 만들기 위해 LFSR의 각 비트값과 비트별 XORing할 값이다.

④ LFSR과 비트별 XORing할 값이 일정한 패턴을 따라가는지 파악

위의 과정을 통해 LFSR의 값은 패턴카운터의 5비트 카운터의 값과 선택적으로 XORing되어 원하는 결정 패턴을 패턴카운터/비트카운터로부터 스캔체인으로 삽입할 수 있도록 한다. 이 때, LFSR의 값과 비트별 XORing할 5비트의 각 비트값이 연속적으로 하나의 값으로 고정되어 있다든지, 매번 값이 플립된다든지, 2번에 한번씩 값이 플립된다든지, 혹은, 4번,8번에 한번씩 값이 플립된다든지 하는 특징이 있다면 그 특징을 파악한다.

<그림 8>에서 음영 처리된 값을 보면, 00001, 00011, 00000인데, 이 값들은 다섯 번째 비트가 1로 시작해서 2번에 한번씩 값이 플립핑되고, 네 번째 비트가 매번 플립핑되는 특징을 갖고 있다. 따라서, 이 경우 멀티플렉서의 컨트롤 신호인 LFSR의 다섯 번째 비트를 비트 카운터의 네 번째 비트와 XORing하고, LFSR의 네 번째 비트를 비트 카운터의 다섯 번째 비트와 XORing하고, 비트 카운터의 시드값을 1로 하고, LFSR의 나머지 비트는 다른 것들과 XORing하지 않은 채로 두면, 원하는 결정 패턴을 얻을 수 있다. 지금의 경우 뿐 아니라,

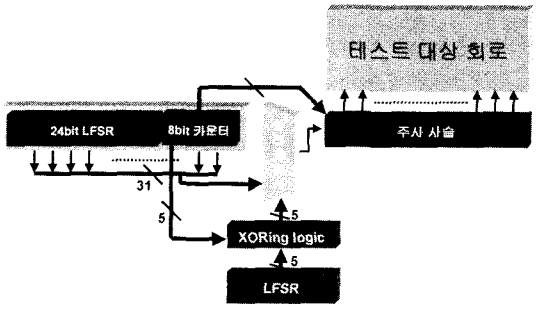


그림 5. 제안하는 결정 패턴 생성기의 구조
Fig. 5. The proposed deterministic pattern generator.

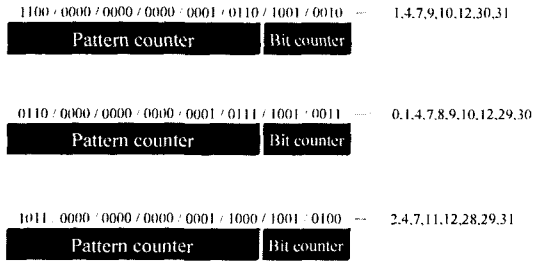


그림 6. 패턴카운터/비트카운터에서 결정 패턴의 비트와 같은 값을 갖는 위치 파악
Fig. 6. Identification of the cells matching the required deterministic pattern bit.

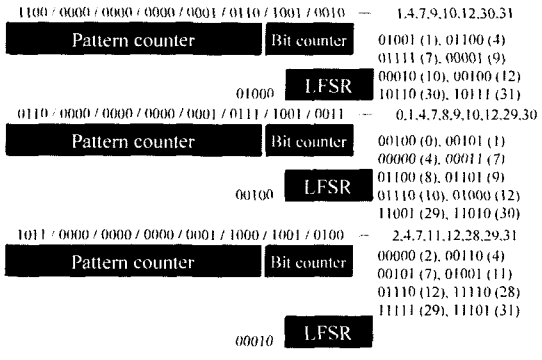


그림 7. LFSR의 상태값과 비트별 XOR할 값 파악
Fig. 7. Identification of the value to be XORed with LFSR state.

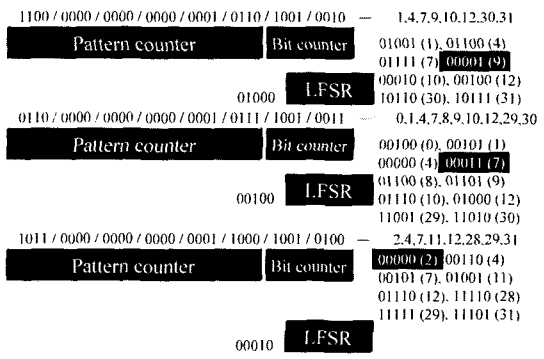


그림 8. LFSR와 비트별 XORing할 값의 특성 파악
Fig. 8. To check which path is longest.

매 시점마다, 다양한 경우가 존재하므로, 이 경우 말고도 다른 구성을 통해 결정 패턴을 구할 수 있다. 따라서, 하나의 구성으로 가장 긴 패턴동안 결정 패턴과 매치되는 구성을 선택하고, 그 다음에도 다른 하나의 구성으로 가장 긴 패턴동안 결정 패턴과 매치되는 구성을 선택하는 방식으로, 모든 결정 패턴이 생성될 때까지의 결정 패턴 생성 로직을 구성한다. 이 방법은 매우 다양한 결정 패턴 생성 로직 구성 방법 중 가장 효율적인 구성 방법을 선택할 수 있으므로 그만큼 간단한 결정 패턴 생성 로직을 통하여, 모든 결정 패턴을 생성할 수 있는 장점이 있다.

⑤ seed++

위의 과정이 끝나면, 시드값을 하나 올려서, 같은 과정을 반복한다.

⑥ 시드값이 seed = 31이 될 때까지 ② ~ ⑤의 과정을 반복한다.

이러한 과정이 끝나면, 시드값이 0일 때부터, 31일 때까지의 결과를 비교하여, 가장 작은 수의 XORing 로직 변경이 필요한 시드값을 선정하여, 결정 패턴 생성 로직을 구성한다.

Ⅲ. 실험결과

연구의 진행과 마찬가지로 실험결과도 의사 무작위 패턴 생성을 위한 실험결과와 결정 패턴 생성을 위한 실험결과 두 부분으로 나누었다.

1. 의사 무작위 패턴 생성을 위한 실험결과

이번 실험은 ISCAS85 와 ISCAS89의 조합회로 및 순차회로를 대상으로 하였고, 그 결과는 아래의 <표 2>, <표 3>과 같다.

<표 2>는 4개의 섹션으로 나뉘어 있는데, 첫 번째와 두 번째 섹션은 대조구로서, 첫 번째 섹션은 기존의 32비트 LFSR을 사용한 결과이고, 두 번째 섹션은 세 번째, 네 번째와 동일한 하드웨어를 소모하는 크기의 패턴 생성기를 사용한 결과이다. 세 번째 섹션은 비트 카운터를 LFSR 형태로 구현한 것이고, 네 번째 섹션은 비트 카운터를 카운터 형태로 구현한 것이며, 세 번째, 네 번째 모두 5비트 LFSR을 사용한 결과이다. 각각의 섹션은 검출고장수와 고장검출율로 이루어져 있는데, 이는 각각 모든 고장 중 검출되지 않은 고장의 수와 의사 무작위 테스트의 고장검출율을 나타낸다. <표 2>

표 2. 작은 크기의 LFSR로 구한 고장검출율
Table 2. Fault coverages with smaller LFSR.

회로	32비트 LFSR (기존)		12비트 LFSR (기존)		5비트 LFSR (LFSR)		5비트 LFSR (카운터)	
	남은 고장수	고장 검출율	남은 고장수	고장 검출율	남은 고장수	고장 검출율	남은 고장수	고장 검출율
c432	4	99.24	4	99.24	4	99.24	4	99.24
c499	8	98.94	8	98.94	8	98.94	8	98.94
c880	23	97.56	34	96.39	13	98.62	21	97.77
c1355	16	98.98	13	99.17	13	99.17	10	99.36
c1908	19	98.99	25	98.67	16	99.15	18	99.04
c2670	429	84.38	435	84.16	432	84.27	432	84.27
c3540	146	95.74	157	95.42	158	95.39	148	95.68
c5315	63	98.82	68	98.73	66	98.77	64	98.80
c6288	34	99.56	34	99.56	34	99.56	34	99.56
c7552	491	93.50	543	92.81	460	93.91	465	93.84
s208	5	97.67	5	97.67	4	98.14	3	98.60
s344	0	100.00	0	100.00	0	100.00	0	100.00
s349	2	99.43	2	99.43	2	99.43	2	99.43
s382	0	100.00	0	100.00	0	100.00	0	100.00
s386	1	99.74	36	90.62	2	99.48	12	96.88
s400	6	98.58	6	98.58	6	98.58	6	98.58
s420	44	89.77	67	84.42	33	92.33	35	91.86
s444	14	97.05	14	97.05	14	97.05	14	97.05
s510	0	100.00	0	100.00	0	100.00	0	100.00
s526	16	97.12	13	97.66	15	97.30	18	96.76
s641	12	97.43	19	95.93	22	95.29	20	95.72
s1196	57	95.41	52	95.81	72	94.20	51	95.89
s1238	134	90.11	126	90.70	132	90.26	129	90.48
s1423	50	96.70	162	89.31	26	98.28	26	98.28
s1488	14	99.06	28	98.12	27	98.18	24	98.38
s1494	26	98.27	41	97.28	33	97.81	36	97.61
s5378	158	96.57	146	96.83	131	97.15	134	97.09
s13207	857	91.27	2562	73.90	444	95.48	602	93.87
s15850	1025	91.26	1958	83.30	1034	91.18	1121	90.44
s38417	2156	93.09	4526	85.48	2424	92.23	2043	93.45
s38584	2028	94.41	2613	92.80	1946	94.64	1922	94.71

에서도 알 수 있듯이, 5비트의 LFSR을 사용한 것이 고장검출율이 평균적으로 비슷하거나 약간 우위에 있는 것을 알 수 있다. 또한, 12비트 LFSR을 사용한 기존 방법의 경우, 큰 회로에 대해서는 고장검출율이 현저하게 떨어지는 것을 볼 수 있다. 여기서 기존 방법의 12비트 LFSR을 사용한 경우를 비교한 이유는 이를 위한 하드웨어 오버헤드가 새로 제안한 방법의 5비트 LFSR과 멀티플렉서를 사용하는 경우의 하드웨어 오버헤드와 동일하다는 가정을 했기 때문이다. 따라서, 기존 방법을 사용한 32비트 LFSR의 경우는 새로 제안한 방법보다 약 20개의 cell을 더 갖는 하드웨어의 부담을 갖는다고 할 수 있다. 이와같이, 5비트의 LFSR을 사용한 새로운 구조는 훨씬 큰 하드웨어를 사용하는 기존의 방법에 비해 하드웨어를 덜 쓰면서, 의사 무작위 패턴 테스트에서 비슷하거나 더 높은 고장검출율을 보인다는 것을 알 수 있다. 의사 무작위 테스트에서는 매우

효율적인 LFSR을 사용하고 있지만, 기존의 하드웨어도 패턴 생성에 이용함으로써 더 작은 하드웨어를 사용하고도 같은 성능의 패턴 생성을 할 수 있게 되었다.

<표 3>은 고장검출율 외에 또 다른 성능평가 항목인 테스트 시간을 검증한 것으로 기존의 방법에 비해 더 많은 패턴이 필요하지 않음을 알 수 있으므로, 새로운 구조의 효율성을 입증하고 있다. <표 3>에서는 <표 2>에 있는 기존의 방법의 12비트 LFSR을 제외하였는데, 이는 이미 <표 2>에서 새로 제안한 방법이 기존의 12비트 LFSR을 사용한 방법보다 우수함은 보였으므로, <표 3>에서는 그보다 훨씬 큰 하드웨어를 사용하는 기존 방법의 32비트 LFSR과 새로 제안하는 구조의 고장검출율을 구하는데까지의 가한 패턴 수를 비교하여, 새로 제안한 방법에서 생성되는 패턴의 다양함을 보이려 한 것이다.

표 3. 작은 크기의 LFSR로 테스트하는데 걸린 총 패턴 수

Table 3. The total number of patterns taken by tests with smaller LFSR.

circuits	32비트 LFSR (기존)			5비트 LFSR (LFSR)			5비트LFSR (카운터)		
	남고장수	얻고검출율	총패턴수	남고장수	얻고검출율	총패턴수	남고장수	얻고검출율	총패턴수
c432	4	99.24	42624	4	99.24	43776	4	99.24	47232
c499	8	98.94	48544	8	98.94	52480	8	98.94	51168
c880	23	97.56	90240	13	98.62	140160	21	97.77	90240
c1355	16	98.98	83656	13	99.17	101024	10	99.36	86592
c1908	19	98.99	88704	16	99.15	103488	18	99.04	106656
c2670	429	84.38	350432	432	84.27	812704	432	84.27	350432
c3540	146	95.74	163200	158	95.39	131200	148	95.68	179200
c5315	63	98.82	324672	66	98.77	364544	64	98.80	273408
c6288	34	99.56	35840	34	99.56	35840	34	99.56	35840
c7552	491	93.50	874368	460	93.91	854496	465	93.84	1033344
s208	5	97.67	24320	4	98.14	29792	3	98.60	21888
s344	0	100.00	6144	0	100.00	8448	0	100.00	3072
s349	2	99.43	26112	2	99.43	26880	2	99.43	26880
s382	0	100.00	10752	0	100.00	10752	0	100.00	10752
s386	1	99.74	42848	2	99.48	20384	12	96.88	20800
s400	6	98.58	30720	6	98.58	28416	6	98.58	26880
s420	44	89.77	77280	33	92.33	43680	35	91.86	84000
s444	14	97.05	37632	14	97.05	33024	14	97.05	28416
s510	0	100.00	16800	0	100.00	19200	0	100.00	17600
s526	16	97.12	39936	15	97.30	36864	18	96.76	34560
s641	12	97.43	86400	22	95.29	74304	20	95.72	69120
s1196	57	95.41	98304	72	94.20	89088	51	95.89	94208
s1238	134	90.11	98304	132	90.36	125952	129	90.48	97280
s1423	50	96.70	131040	26	98.28	224224	26	98.28	206752
s1488	14	99.06	30016	27	98.18	26880	24	98.38	26432
s1494	26	98.27	30016	33	97.81	27776	36	97.61	26432
s5378	158	96.57	540992	131	97.15	979264	134	97.09	862848
s13207	857	91.27	6496000	444	95.48	10841600	602	93.87	8220800
s15850	1025	91.26	5474560	1034	91.18	3284736	1121	90.44	2502656
s38417	2156	93.09	19701760	2424	92.23	28434432	2043	93.45	21512192
s38584	3028	94.41	13773312	1946	94.64	18270720	1922	94.71	14241792

2. 결정 패턴 생성을 위한 실험결과

결정 패턴 생성에서의 결과 비교 결과는 <표 4>에 요약되어 있다. <표 4>에서 바꿔야하는 bit의 수란 그 중 제안한 구조의 환경에서 멀티플렉서의 컨트롤 신호를 만들기 위해 LFSR의 값과 비트별 XORing logic을 바꾸어야 하는 수를 말하며, 이 수가 결정 패턴 생성을 위한 디코딩 로직의 크기와 직결되어 있다.

표 4. specified bit의 수와 제안한 구조에서의 비트 플립을 해야하는 bit의 수 비교

Table 4. Comparison between the number of specified bits and the bits that should be flipped.

회 로	specified bit[1]의 수	바꿔야하는 bit의 수	회 로	specified bit의 수	바꿔야하는 bit의 수
c880	213	34	cs820	175	31
c1908	69	14	cs832	163	26
c2670	2742	373	cs838	2119	271
c3540	99	16	cs953	155	19
c5315	169	27	cs1196	454	53
c7552	4757	625	cs1238	442	50
cs208	23	7	cs1423	190	29
cs386	20	2	cs1488	42	8
cs420	426	58	cs1494	21	3
cs526	173	23	cs5378	1174	153
cs641	302	51	cs9234	9837	1299
cs713	302	49			

<표 4>에서 보면 알 수 있듯이 바꿔야 하는 bit의 수가 평균적으로 specified bit의 수의 13% 정도인 것을 확인할 수 있으며, 이는 위의 알고리즘이 사용되지 않은 상태에서 결정 패턴을 생성할 때의 확률적인 기대값인 50%보다 훨씬 낮은 값임을 또한 확인할 수 있다. 또한, 이같은 결과는 결정 패턴 생성을 위한 로직의 크기도 25%(13/50)정도로 기존의 것보다 작아질 수 있음을 의미한다.

Ⅳ. 결 론

본 논문에서는 패턴 카운터와 비트 카운터를 효율적으로 패턴 생성에도 가담시킴으로써, 의사 무작위 테스트에서 고장검출율이나 테스트 시간에 있어서의 성능저하 없이 테스트 하드웨어를 절약할 수 있었다.

또한, 새로운 결정 패턴 생성 알고리즘을 개발함으로써, 테스트 하드웨어를 더욱 더 줄일 수 있었다. 이 알

1) specified bit의 수란 결정 패턴을 위한 벡터중 don't care 값이 아닌 비트의 총 개수이다.

고리즘은 결정 패턴의 생성을 위해 카운터와 LFSR의 값을 XORing하는 방법을 쓰는데, XORing 연결의 변경 횟수를 최소화하는데 초점을 맞추고 있다. 그 이유는 XORing 연결의 변경 횟수가 결정 패턴을 위한 전체 로직의 크기를 결정하기 때문이다.

실험은 ISCAS85 회로와 ISCAS89 회로에 적용하였으며, 고장검출율과 테스트 시간, 그리고 XORing 연결의 변경 횟수로서 본 논문에서 제안하는 방법의 성능을 확인하였다.

참 고 문 헌

[1] M. Abramovici, M. A. Breuer and A. D. Fiedman, Digital Systems Testing and Testable Design, Computer Science Press, 1990.

[2] B. Pouya and N. A. Touba, "Synthesis of zero-aliasing elementary-tree space compactors", in Proc. IEEE VLSI Test Symp., 1998, pp. 70~77.

[3] N. A. Touba and E. J. McCluskey, "Bit-fixing in pseudorandom sequences for scan BIST", IEEE Trans. Computer-Aided Design, vol. 20, 2001, pp. 545~555.

[4] G. Kiefer and H. J. Wunderlich, "Using BIST Control for Pattern Generation", in Proc. Int. Test Conf., 1997, pp. 347~355.

[5] K. Chakrabarty, B. T. Murray and V. Iyengar, "Deterministic Built-In Self Test Pattern Generation for High-Performance Circuits using Twisted-Ring Counters", IEEE Trans. VLSI Systems, vol. 8, no. 5, 2000, pp. 633~636.

[6] C. Fagot, P. Girard and C. Landrault, "On Using Machine Learning for Logic BIST", in Proc. Int. Test Conf., 1997, pp. 338~346.

[7] S. Hellebrand, H. G. Liang and H. J. Wunderlich, "A Mixed Mode BIST Scheme Based On Reseeding of Folding Counters", in Proc. Int. Test Conf., 2000, pp. 778~784.

[8] G. Kiefer, H. Wunderlich: "Deterministic BIST with Multiple Scan Chains", in Proc. Int. Conf., 1998, pp. 1057~1064.

[9] C. Fagot, O. Gascuel, P. Girard, C. Landrault, "On Calculating Efficient LFSR Seeds for Built-In Self Test", Test Workshop. European, 1999, pp. 7~14.

[10] N. A. Touba, E. J. McCluskey, "Altering a pseudo-random bit sequence for scan-based BIST", in Proc. Int. Test Conf., 1996, pp. 167~175.

[11] S. B. Akers and W. Jansz, "Test Set Embedding in Built-In Self-Test Environment", in Proc. Int. Test Conf., 1989, pp. 257~263.

[12] S. Hellebrand, H.-J. Wunderlich, O. F. Haberl, "Generating Pseudo-Exhaustive Vectors for External Testing", in Proc. Int. Test Conf., 1990, pp. 670~679.

[13] H. S. Kim, J. K. Lee, S. H. Kang, "A new multiple weight set calculation algorithm", in Proc. Int. Test Conf., 2001, pp. 878~884.

[14] S. Wang, "Low hardware overhead scan based 3-weight weighted random BIST", Proc. Int. Test Conf., 2001, pp. 868~877.

[15] H. Lee, S. Kang, "A new weight set generation algorithm for weighted random pattern generation," in Proc. Int. Test Conf., 1999, pp. 160~165.

[16] C. V. Krishna, A. Jas, N. A. Touba, "Test vector encoding using partial LFSR reseeding", in Proc. Int. Test Conf., 2001, pp. 885~893.

[17] E. Kalligeros, X. Kavousianos, D. Bakalis, D. Nikolos, "New reseeding technique for LFSR-based test pattern generation", On-Line Testing Workshop, 2001, pp. 80~86.

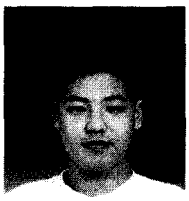
[18] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, B. Courtois, "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers", IEEE Trans. Comput., 1995, pp. 223~233.

[19] S. Hellebrand, S. Tarnick, J. Rajski, B. Courtois, "Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers", in Proc. Int. Test Conf., 1992, pp. 120~129.

저 자 소 개

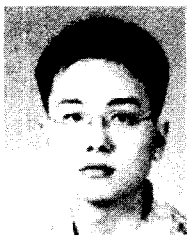
金 玄 炖(正會員)

2001년 2월 : 연세대학교 전기공학과 졸업. 2003년 2월 : 연세대학교 전기전자공학과 졸업(석사). 현재 : 삼성전자 시스템 LSI 사업부 SOC연구소



金 容 準(正會員)

2002년 2월 : 연세대학교 전기공학과 졸업. 현재 : 연세대학교 전기전자공학과 석사과정.



愼 鏞 升(正會員)

2002년 2월 : 연세대학교 전기공학과 졸업. 현재 : 연세대학교 전기전자공학과 석사과정

姜 成 昊(正會員)

1986년 2월 : 서울대 공대 제어계측공학과 졸업. 1988년 5월 : The University of Texas at Austin 전기 및 컴퓨터공학과 졸업(석사). 1992년 5월 : The University of Texas at Austin 전기 및 컴퓨터공학과 졸업(공박). 미국 Schlumberger 연구원. Motorola 선임 연구원. 현재 : 연세대학교 공과대학 전기전자공학과 부교수